Ciências ULisboa | Informática

Aplicações Distribuídas

2021/22

Projeto 1

1. Descrição geral

A avaliação prática da disciplina de Aplicações Distribuídas está divida em quatro projetos. Os Projetos 1 e 2 têm continuidade entre eles e, por essa razão, **é muito importante que consigam cumprir os objetivos do Projeto 1, de forma a não tornar mais difícil o Projeto 2**.

O objetivo geral do projeto será concretizar um gestor de bloqueios a recursos para leituras e escritas (*Read-Write Locks*). O seu propósito é controlar o acesso a um conjunto de recursos partilhados num sistema distribuído, onde diferentes clientes podem requerer o acesso aos mesmos de forma concorrente. Um recurso pode ser bloqueado para a escrita exclusivamente por um só cliente de cada vez, até um máximo de `K` bloqueios de escrita ao longo do tempo. Ou seja, findo os `K` bloqueios de escrita permitidos, o recurso fica desabilitado (i.e., indisponível para novos bloqueios). Um recurso pode ser bloqueado para a leitura por um número qualquer de clientes, caso o recurso não esteja bloqueado para a escrita ou desabilitado. Basicamente, o gestor previne conflitos entre escritas de clientes diferentes, assim como previne conflitos entre operações de leitura e escrita enquanto o recurso está ativo. O gestor será concretizado num servidor escrito na linguagem *Python 3*. A **Figura 1** ilustra a arquitetura a seguir no servidor de bloqueios (*lock_server*), bem como a estrutura de dados em *Python* que o suporta.

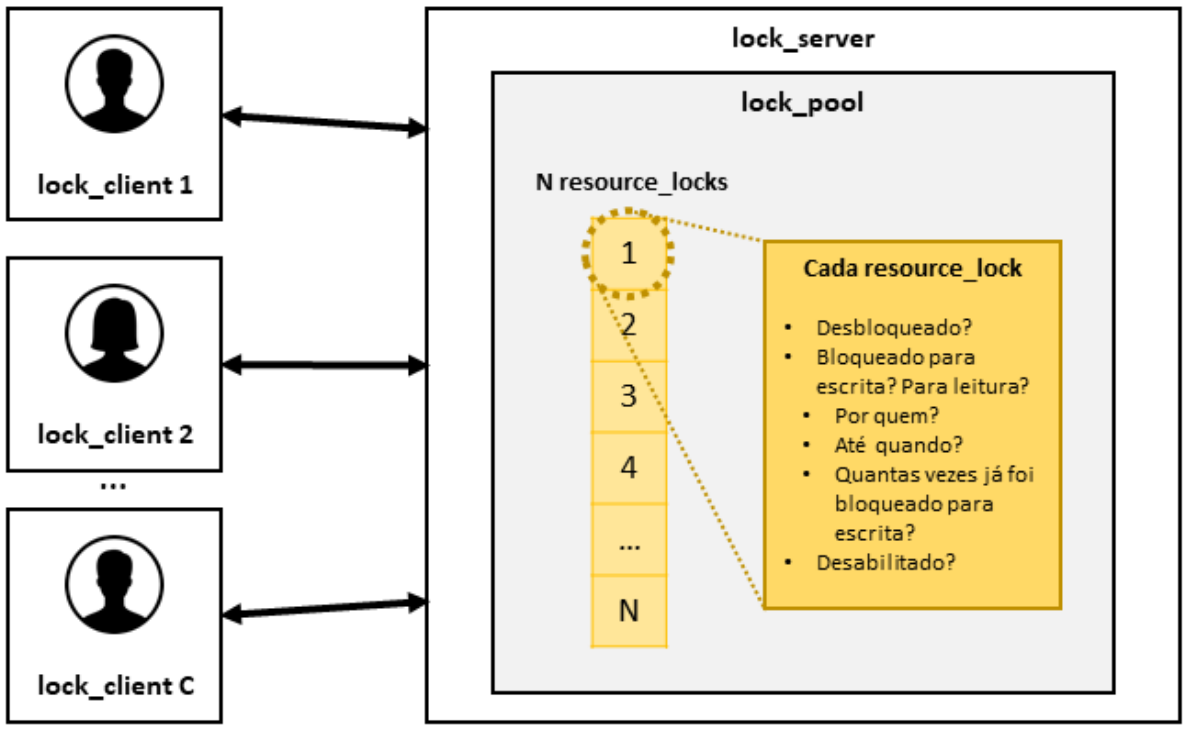

Figura 1– Arquitetura do servidor de bloqueios e estrutura de dados através de duas classes.

Nesta primeira fase, o servidor aceita ligações de clientes, recebe e processa os pedidos, responde ao cliente, e termina a ligação. O cliente efetua o pedido de ligação ao servidor, envia o pedido ao servidor, recebe a resposta e termina a ligação.

Entre outros aspetos, na segunda fase vamos considerar a possibilidade de o servidor atender a múltiplos clientes em simultâneo e de o cliente não necessitar de estabelecer e terminar a ligação sempre que pretende enviar um novo comando.

2. Descrição específica

A resolução do Projeto 1 está dividida em três passos:

1. Definição do formato das mensagens a serem trocadas entre o cliente e o servidor;
2. Definição da estrutura de dados a ser mantida pelo servidor;
3. Concretização dos processos no cliente e no servidor.

2.1. Comandos suportados e formato das mensagens

O sistema será controlado através de 7 comandos possíveis. Uma sequência de comandos compõe uma história de execução que será apresentada para o cliente. O programa cliente fará uma primeira validação do nome de cada comando fornecido e enviará para o servidor os comandos pertinentes através de *strings* com um formato de mensagem específico. O servidor, por sua vez, validará o comando e os argumentos recebidos na mensagem, fará o respetivo processamento e responderá ao cliente também com uma *string* que indica o resultado do processamento do comando.

Dos 7 comandos previstos, 5 deles (listados na Tabela 1) resultarão em envios de mensagens do cliente para o servidor e em respostas do servidor para o cliente. Os restantes 2 comandos serão apresentados mais a frente.

Tabela 1 - Lista de comandos suportados pelo servidor e formato das mensagens de pedido e resposta.

Comando	**Comando recebido pelo cliente**	**Formato da *string* enviada pelo cliente**	**Resposta do servidor**
LOCK	LOCK <R|W> <número do recurso> <limite de tempo>	LOCK <R|W> <número do recurso> <limite de tempo> <id do cliente>	OK ou NOK ou UNKNOWN RESOURCE
UNLOCK	UNLOCK <R|W> <número do recurso>	UNLOCK <R|W> <número do recurso> <id do cliente>	OK ou NOK ou UNKNOWN RESOURCE
STATUS	STATUS <número do recurso>	STATUS <número do recurso>	LOCKED-W ou LOCKED-R ou UNLOCKED ou DISABLED ou UNKNOWN RESOURCE
STATS	STATS K <número do recurso>	STATS K <número do recurso>	<número de bloqueios de escrita feitos no recurso> ou UNKNOWN RESOURCE
	STATS N	STATS N	<número de recursos UNLOCKED>
	STATS D	STATS D	<número de recursos DISABLED>
PRINT	PRINT	PRINT	<estado de todos os recursos>

Todos os recursos do servidor iniciam a sua execução com o estado `UNLOCKED`, o contador de bloqueios de escrita com o valor `0` (zero) e as listas de bloqueios de escrita e leitura vazias.

O comando **`LOCK`** bloqueia um determinado recurso para leitura (`R`) ou escrita (`W`) durante um tempo de concessão específico (em segundos) para o cliente que está a enviar o pedido. Se o pedido for para um bloqueio de escrita (`W`), o recurso pode ser bloqueado para a escrita apenas se ele estiver no estado `UNLOCKED` e o limite de bloqueios de escrita `K` do recurso não tiver sido atingido. Neste caso, o servidor deve passar o recurso para o estado `LOCKED-W`, incrementar o contador de bloqueios de escrita, calcular o limite de tempo de concessão (i.e., *deadline = tempo atual + tempo de concessão*), registar o tuplo `(id do cliente, deadline)` na lista de bloqueios de escrita e retornar `OK`. Caso o recurso esteja `LOCKED-W`, `LOCKED-R` ou `DISABLED`, o servidor deve retornar `NOK` ao pedido de bloqueio para a escrita.

Se o pedido for para um bloqueio de leitura (`R`), o recurso pode ser bloqueado para a leitura apenas se ele estiver no estado `LOCKED-R` ou `UNLOCKED`. Neste caso, o servidor deve calcular o limite de tempo de concessão (i.e., novamente, *deadline = tempo atual + tempo concessão*), registar o tuplo `(id do cliente, deadline)` na lista de bloqueios de leitura, passar o estado do recurso para `LOCKED-R` (caso ainda não esteja) e retornar `OK`. Caso o recurso esteja no estado `LOCKED-W` ou `DISABLED`, o servidor deve retornar `NOK`.

Em ambos casos (i.e., bloqueios de leitura e escrita), se o pedido se referir a um recurso inexistente (i.e., um número de recurso menor que `1` ou maior que `N`), o servidor deverá retornar `UNKNOWN RESOURCE`.

O comando **`UNLOCK`** remove um bloqueio de leitura (`R`) ou escrita (`W`) registado num determinado recurso para o cliente que está a enviar o pedido. Se o pedido for um desbloqueio de escrita, o recurso solicitado estiver no estado `LOCKED-W` e o cliente que está a bloquear a escrita neste recurso for o mesmo que o que está a pedir o desbloqueio, então o servidor deve desbloquear a escrita no recurso (através da remoção do respetivo tuplo da lista de bloqueios de escrita e a passagem do recurso para o estado `UNLOCKED`) e retornar `OK`. Caso o recurso esteja no estado `UNLOCKED` ou `DISABLED` ou ainda se ele estiver bloqueado para a escrita (`LOCKED-W`) por outro cliente, o servidor deve retornar `NOK`.

Se o pedido for um desbloqueio de leitura, o recurso solicitado estiver no estado `LOCKED-R` e o cliente que está a pedir o desbloqueio pertencer à lista de bloqueios de leitura, então o servidor deve remover o respetivo tuplo da lista de bloqueios de leitura, verificar se não há outros clientes com bloqueios de leitura (neste caso, passar o recurso ao estado `UNLOCKED`) e retornar `OK`. Caso o recurso esteja no estado `LOCKED-W` ou `UNLOCKED`, ou ainda o cliente que está a pedir o desbloqueio de leitura não pertencer à lista de clientes com bloqueios de leitura ativos, o servidor deve retornar `NOK`.

Por fim, se o pedido de desbloqueio (seja de leitura ou de escrita) referir-se a um recurso inexistente (e.g., um número de recurso menor que `1` ou maior que `N`), o servidor deverá retornar `UNKNOWN RESOURCE`.

Note que segundo a Tabela 1, tanto no `LOCK` quanto no `UNLOCK` a diferença entre o comando apresentado ao cliente e a mensagem enviada ao servidor é o `<id do cliente>`, o qual cada

programa cliente é responsável por inserir nos seus pedidos. Todos os outros comandos possuem o mesmo formato no comando e no pedido.

O comando **STATUS** é utilizado para obter o estado atual de um determinado recurso. O servidor retorna o estado do recurso solicitado (i.e., `UNLOCKED`, `LOCKED-W`, `LOCKED-R` ou `DISABLED`). Se o pedido se referir a um recurso inexistente (e.g., um número de recurso menor que `1` ou maior que `N`), o servidor deverá retornar `UNKNOWN RESOURCE`.

O comando **STATS** é utilizado para obter outras informações sobre o servidor de bloqueios e possui 3 formas. Na primeira (`STATS K <número do recurso>`), o servidor retorna o número de bloqueios de escrita realizados no recurso especificado. Na segunda (`STATS N`), o servidor retorna o número total de recursos que se encontram disponíveis atualmente (i.e., no estado `UNLOCKED`). Na terceira (`STATS D`), o servidor retorna o número total de recursos que se encontram desabilitados atualmente (i.e., no estado `DISABLED`).

O comando **PRINT** é utilizado para obter uma visão geral do estado do serviço de gestão de bloqueios, onde o servidor retorna uma *string* que contem os seguintes campos (separados por espaços), numa linha para cada recurso: a letra `R` para indicar um recurso, o id do recurso, o seu estado e o número de bloqueios de escrita efetuados no mesmo. Se o recurso estiver no estado `LOCKED-W`, a linha do recurso ainda incluirá o id do cliente que detém o bloqueio de escrita no recurso e o limite de tempo de concessão (i.e., *deadline*) que indica o término do bloqueio de escrita deste recurso (em segundos desde a época, ver a Secção 2.3.3). Se o recurso estiver no estado `LOCKED-R`, a linha do recurso também incluirá o número de clientes que possuem bloqueio de leitura ativos no recurso e o maior limite de tempo de concessão (i.e., *deadline*), o qual indica o término de todos os bloqueios de leitura deste recurso (em segundos desde a *época*, ver a Secção 2.3.3). Um exemplo de resultado do comando `PRINT` de um servidor com 4 recursos pode ser o seguinte:

```
R 1 UNLOCKED 1
R 2 DISABLED 3
R 3 LOCKED-W 2 5 1612873214
R 4 LOCKED-R 3 6 1612873387
```

O programa cliente será o responsável por verificar se o comando apresentado existe, se não possui gralhas no seu nome e se não faltam argumentos conforme a especificação. Caso o comando não exista ou possua gralhas, o cliente apresentará o resultado "`UNKNOWN COMMAND`" e não enviará o pedido ao servidor. Caso faltem argumentos, o cliente apresentará o resultado "`MISSING ARGUMENTS`" e não enviará o pedido ao servidor.

Os 2 comandos que serão tratados apenas do lado do cliente (i.e., não são enviados pedidos para o servidor) são:

Tabela 2 - Lista de comandos suportados apenas pelo cliente.

Comando	Comando recebido pelo cliente
SLEEP	SLEEP <limite de tempo>
EXIT	EXIT

O comando **SLEEP** faz com que o cliente adormeça durante um determinado tempo (em segundos). Ou seja, o cliente esperará este tempo antes de interpretar o próximo comando.

O comando **EXIT** encerra a execução do cliente. Pode-se assumir que todas as execuções terão um comando `EXIT` no final.

Seguem alguns exemplos de sequências de comandos que poderão ser utilizados para analisar a execução dos programas a serem desenvolvidos:

Tabela 3 – Exemplos de sequências de comandos. Os comandos do quarto exemplo contém erros ou argumentos em falta e não deveriam ser enviados ao servidor. Todos os comandos do quinto exemplo utilizam recursos inexistentes (e.g., para `N=3`).

`LOCK-W 0 30`	`LOCK-W 0 10`	`LOCK-W 0 10`	`BLOCK-W 0 10`	`LOCK-W 3 10`
`SLEEP 5`	`LOCK-W 1 10`	`LOCK-R 0 10`	`UNBLOCK-W 0`	`LOCK-W -1 10`
`STATUS 0`	`LOCK-W 2 10`	…	`LOCK-R`	`LOCK-R 1000 10`
`STATS K 0`	`LOCK-R 3 10`	`LOCK-R 0 10`	`LOCK-R 0`	`LOCK-R 3 10`
`STATS N`	`LOCK-R 4 10`	`UNLOCK-W 0`	`UNLOCK-R`	`UNLOCK-W -2`
`UNLOCK-W 0`	`STATS N`	`STATUS 0`	`STATS X`	`STATUS 1000`
`PRINT`	`SLEEP 20`	`STATS K 0`	`STATUS`	`STATS K -2`
`EXIT`	`PRINT`	`STATS D`	`EXIT`	`EXIT`
	`EXIT`	`EXIT`		

2.2. Estrutura de dados

Os dados a guardar no servidor consistem de informação relacionada com a gestão dos bloqueios no acesso concorrente a um conjunto de **N** recursos partilhados. Serão utilizadas duas classes em Python [1] para estruturar a informação e o acesso à mesma.

Sobre cada recurso dever-se-á saber a seguinte informação:

- O estado do recurso (p. ex., se está bloqueado para a leitura, bloqueado para a escrita, desbloqueado ou desabilitado);
- Quantas vezes já foi bloqueado para a escrita;
- No caso de estar bloqueado para a escrita:
  - qual o cliente que detém a concessão de bloqueio de escrita; e
  - até quando essa concessão é válida.
- No caso de estar bloqueado para a leitura:
  - quais os clientes que detêm as concessões de bloqueio de leitura; e
  - até quando cada uma dessas concessões são válidas.

A definição de um recurso deverá ser implementada na classe *resource_lock*, mostrada de seguida. Esta classe deve ser definida e instanciada no ficheiro `lock_server.py` fornecido para a implementação do servidor.

```
class resource_lock:
    def __init__(self, resource_id):
        pass # Remover esta linha e fazer implementação da função

    def lock(self, type, client_id, time_limit):
        pass # Remover esta linha e fazer implementação da função

    def release(self):
        pass # Remover esta linha e fazer implementação da função

    def unlock(self, type, client_id):
        pass # Remover esta linha e fazer implementação da função

    def status(self, option):
        pass # Remover esta linha e fazer implementação da função

    def disable(self):
        pass # Remover esta linha e fazer implementação da função

    def __repr__(self):
        output = ""
        # Se o recurso está bloqueado para a escrita:
        # R <num do recurso> LOCKED-W <vezes bloqueios de escrita> <id do
cliente> <deadline do bloqueio de escrita>
        # Se o recurso está bloqueado para a leitura:
        # R <num do recurso> LOCKED-R <vezes bloqueios de escrita> <num
bloqueios de leitura atuais> <último deadline dos bloqueios de leitura>
        # Se o recurso está desbloqueado:
        # R <num do recurso> UNLOCKED
        # Se o recurso está inativo:
        # R <num do recurso> DISABLED
        return output
```

O conjunto de **N** recursos será definido pela classe *lock_pool*, a qual deverá ser definida e instanciada também no ficheiro `lock_server.py` já mencionado. Esta classe serve também de interface para o acesso aos dados relacionados aos bloqueios de leitura e escrita de cada recurso, bem como à gestão dos **K** bloqueios permitidos por recurso.

```
class lock_pool:
    def __init__(self, N, K):
        pass # Remover esta linha e fazer implementação da função

    def clear_expired_locks(self):
        pass # Remover esta linha e fazer implementação da função

    def lock(self, type, resource_id, client_id, time_limit):
        pass # Remover esta linha e fazer implementação da função

    def unlock(self, type, resource_id, client_id):
        pass # Remover esta linha e fazer implementação da função

    def status(self, resource_id):
        pass # Remover esta linha e fazer implementação da função

    def stats(self, option, resource_id):
        pass # Remover esta linha e fazer implementação da função

    def __repr__(self):
        output = ""
        # Acrescentar no output uma linha por cada recurso
        return output
```

2.3. Concretização do cliente e do servidor

2.3.1 Cliente

O cliente, a implementar no ficheiro `lock_client.py` fornecido, deverá receber os seguintes parâmetros pela linha de comandos, pela ordem apresentada:

- o id único do cliente
- o IP ou *hostname* do servidor que fornece os recursos;
- o porto TCP onde o servidor recebe pedidos de ligação;

Desta forma, um exemplo de inicialização do cliente é o seguinte:

```
$  python3 lock_client.py 1 localhost 9999
```

O cliente deverá, de forma cíclica:

1. pedir ao utilizador um comando para enviar ao servidor, através da prompt, "`comando > `"
2. estabelecer a ligação ao servidor;
3. enviar o comando e receber a resposta do servidor;
4. terminar a ligação ao servidor;
5. apresentar a resposta recebida.

A implementação do cliente deverá ser feita com base na classe *server_connection* definida no ficheiro `net_client.py`, fornecido para o efeito. Por sua vez esta classe deverá usar o módulo `sock_utils.py` sugerido nas aulas práticas para a implementação relacionada com *sockets* [2].

```
class server_connection:
    def __init__(self, address, port):
        pass # Remover esta linha e fazer implementação da função

    def connect(self):
        pass # Remover esta linha e fazer implementação da função

    def send_receive(self, data):
        pass # Remover esta linha e fazer implementação da função

    def close(self):
        pass # Remover esta linha e fazer implementação da função
```

2.3.2 Servidor

O servidor, a implementar no ficheiro `lock_server.py` fornecido, deverá receber os seguintes parâmetros pela linha de comandos, pela ordem apresentada:

- IP ou *hostname* onde o servidor fornecerá os recursos;
- porto TCP onde escutará por pedidos de ligação;
- número de recursos que serão geridos pelo servidor (**`N`**);
- número de bloqueios permitidos em cada recurso (**`K`**);

Desta forma, um exemplo de inicialização do servidor é o seguinte:

```
$  python3 lock_server.py localhost 9999 4 3
```

O servidor deverá implementar o seguinte ciclo de operações:

1. aceitar uma ligação;
2. mostrar no ecrã informação sobre a ligação (IP/*hostname* e porto de origem do cliente);
3. verificar se existem recursos com bloqueios de leitura ou escrita cujo tempo de concessão tenha expirado, e remover esses bloqueios nos recursos;
4. verificar se existem recursos que já atingiram os **`K`** bloqueios de escrita permitidos, e desativar estes recursos em caso positivo;
5. receber uma mensagem com um pedido;
6. processar esse pedido;
7. responder ao cliente;
8. fechar a ligação;

2.3.3 Detalhes adicionais de implementação e testes

O tempo limite da concessão de um bloqueio deverá ser manipulado através da função `time()` do módulo *time* do *Python* [3]. Esta devolve o número de segundos desde a *época* (uma data de referência à qual se soma um número de segundos para obter a data e hora locais).

A receção dos comandos no ciclo do programa cliente deverá ser feito através da função `input()`. Esta interrompe a execução do programa e fica à espera de que um novo comando seja apresentado pelo utilizador. A vantagem de utilizar esta função é que ela serve tanto para obter comandos de forma interativa com um utilizador, como receber comandos enviados através do *pipe* para o programa. Para este último, basta gravar a lista de comandos que compõe uma história de execução num ficheiro de texto (e.g., `comandos.txt`) e executar o seguinte comando:

```
$ cat comandos.txt | python3 lock_client.py 1 localhost 9999
```

Para além disso, tanto o programa cliente quanto o servidor devem imprimir todas as mensagens enviadas e recebidas. No caso do cliente, serão todos os pedidos enviados e respostas recebidas. No caso do servidor, serão todos os pedidos recebidos e as respostas a serem enviadas. O esperado é que a saída do cliente seja muito semelhante à saída do servidor, ao ponto que permita fazer a comparação com a ferramenta `diff`.

Por fim, um exemplo de teste será executar as seguintes linhas na consola:

```
$ python3 lock_server.py localhost 9999 4 3 > output_server.log
$ cat commandos.txt | python3 lock_client.py 1 localhost 9999 > output_client.log
$ diff output_server.log output_client.log
```

3. Avaliação

3.1. Entrega

A entrega do Projeto 1 consiste em colocar todos os ficheiros `.py` do projeto numa diretoria cujo nome deve seguir exatamente o padrão `grupoXX` (onde `XX` é o número do grupo). Juntamente com os ficheiros `.py` deverá ser enviado um ficheiro de texto `README.txt` (é em TXT, não é PDF, RTF, DOC, DOCX, etc.) onde os alunos podem relatar a informação que acharem pertinente sobre a sua implementação do projeto (por exemplo, limitações). A diretoria será incluída num ficheiro ZIP cujo nome deve seguir exatamente o padrão `grupoXX.zip` (novamente `XX` é o número do grupo). Esse ficheiro será submetido num recurso a disponibilizar para o efeito na página de AD no moodle da FCUL. Note que a entrega deve conter apenas os ficheiros `.py` e o ficheiro `README.txt`. Qualquer outro ficheiro vai ser ignorado. O não cumprimento destas regras podem anular a avaliação do trabalho.

O prazo de entrega do Projeto 1 é **domingo, 13 de Março de 2022, às 23:59 (WET)**.

3.2. Plágio

Não é permitido aos alunos dos grupos partilharem código com soluções, ainda que parciais, a nenhuma parte do projeto com alunos de outros grupos (nem através do Fórum da disciplina, nem por qualquer outro meio). Além disso, todos os códigos serão testados por um sistema verificador de plágio. Caso alguma irregularidade seja encontrada, os projetos de todos os alunos envolvidos serão anulados e o caso poderá ser reportado aos órgãos responsáveis na Ciências@ULisboa.

4. Bibliografia

[1] https://docs.python.org/3/tutorial/classes.html

[2] https://docs.python.org/3/library/socket.html

[3] https://docs.python.org/3/library/time.html